ENCRYPTION & DECRYPTION OF DATA IN GF (13^n) FIELD

THESIS SUBMITTED BY

ABHISEK PUJARI

ROLL NO. :-10509006

&

CHANDAN PADHI

ROLL NO. :-10509009

UNDER THE GUIDANCE OF

Prof. G.S. RATH

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY, ROURKELA

National Institute of Technology
Rourkela

CERTIFICATE

This is to certify that the thesis entitled **"Encryption and Decryption of Data in GF(13^n) Field"** submitted by **Abhisek Pujari and Chandan Padhi to the Department of Electronics and Communication Engineering** at the **National Institute of Technology, Rourkela**, for the degree of **Bachelor of Technology,** is a record of authentic work carried out under my supervision and guidance. The result presented in this thesis has not been submitted elsewhere for the award of any Degree or Diploma.

This work, in my opinion, has reached the standard of fulfilling the requirements for the award of the degree of **Bachelor of Technology** in accordance with the regulation of the institute.

Date:

Prof. G.S.Rath
Dept. of Electronics &Communication Engg.
National Institute of Technology
Rourkela-769008

ABSTRACT

Cryptography is the science of using mathematics to encrypt and decrypt data. Cryptography enables you to store sensitive information or transmit it across insecure networks (like the Internet) so that it cannot be read by anyone except the intended recipient.
Cryptographic strength is measured in the time and resources it would require to recover the plaintext. The result of *strong cryptography* is cipher text that is very difficult to decipher without possession of the appropriate decoding tool.
A *cryptographic algorithm*, or cipher, is a mathematical function used in the encryption and decryption process. A cryptographic algorithm works in combination with a *key* — a word, number, or phrase — to encrypt the plaintext. The same plaintext encrypts to different cipher text with different keys. The security of encrypted data is entirely dependent on two things: the strength of the cryptographic algorithm and the secrecy of the key.
With growing dependencies on secure information transfer and data security in this information age the requirement for a strong method of securing data is always in demand.
In this project , we have tried to device a new means of securing data by encryption and decryption in GF(13). The reason for using GF(13) is that much research has been done on GF(2) but the GF(13) field is still mostly untouched. Hence there is much scope for designing versatile techniques which will be both secure and faster. This will again reduce the probability of anonymous decryption by undesirable agents. Such technique can be the future of data security with wide application in every field starting with military application, internet security, wireless data transfer and many more.

ACKNOWLEDGEMENTS

On the submission of my thesis report of "Encryption and Decryption of Data in GF (13) field", I would like to extend my gratitude & my sincere thanks to my supervisor Prof**. G.S.Rath**, Department of Electronics and Communication Engineering for his constant motivation and support during the course of my work in the last one year. I truly appreciate and value his esteemed guidance and encouragement from the beginning to the end of this thesis. I am indebted to him for having helped me shape the problem and providing insights towards the solution.

I want to thank all my teachers **Prof. G. Panda**, **Prof. S.K.Patra** and **Prof. S.K. Meher** for providing a solid background for my studies and research thereafter. They have been great sources of inspiration to me and I thank them all from the bottom of my heart.

Above all, I would like to thank all my friends whose direct and indirect support helped me completing my project in time. This thesis would not have been possible without their perpetual moral support.

CONTENTS

1

INTRODUCTION

1.1 CRYPTOGRAPHY

Cryptography is the practice and study of hiding information. It refers to the science and art of transforming messages to make them secure and immune to attacks. It includes encryption and decryption of messages using secret keys. In modern times cryptography is considered a branch of both mathematics and computer science and is affiliated closely with information theory, computer security and engineering. Cryptography is used in applications present in technologically advanced societies; examples include the security of ATM cards, computer passwords and electronic commerce which all depend on cryptography. Before the modern era, cryptography was concerned solely with message confidentiality (i.e., encryption) — conversion of messages from a comprehensible form into an incomprehensible one and back again at the other end, rendering it unreadable by interceptors or eavesdroppers

without secret knowledge (namely the key needed for decryption of that message). In recent decades, the field has expanded beyond confidentiality concerns to include techniques for message integrity checking, sender/receiver identity authentication, digital signatures, interactive proofs and secure computation, among others.

1.2 ENCRYPTION AND DECRYPTION

In cryptography, encryption is the process of transforming information (referred to as plaintext) using an algorithm (called cipher) to make it unreadable to anyone except those possessing special knowledge, usually referred to as a key. The result of the process is encrypted information (referred to as ciphertext). Encryption has long been used by militaries and governments to facilitate secret communication. Encryption is now used in protecting information within many kinds of civilian systems, such as computers, storage devices (e.g. USB flash drives), networks (e.g. the Internet, e-commerce), mobile telephones, wireless microphones, wireless intercom systems, Bluetooth devices and bank automatic teller machines. Encryption is also used in digital rights management to prevent unauthorized use or reproduction of copyrighted material and in software also to protect against reverse engineering. Decryption is the reverse process of encryption. In many contexts, the word encryption also implicitly refers to the reverse process, decryption (e.g. “software for encryption” can typically also perform decryption), to make the encrypted information readable again (i.e. to make it unencrypted).In decryption, the ciphertext is converted into its original format using the key. It is the process of decoding data that has been encrypted into a secret format. Decryption requires a secret key or password.

1.3 NECESSITY OF CRYPTOGRAPHY

To be secured, information need to be hidden from unauthorized access (confidentiality), protected from unauthorized change(integrity), and available to an authorized entity where it is needed(availability). The implementation of these three security goals is made possible through cryptography. Cryptography is the discipline that embodies the principles, means and methods for the transformation of data in order to hide its content, establish its authenticity, prevent its undetected modification or unauthorized use. Cryptographic techniques enable users to protect the privacy of their data (financial or personal records, for example), whether the data are in storage or in transit. Cryptographic techniques also allow users to determine whether someone has altered data —whether, for example, a hacker who has broken into medical records has altered any element of them. They also enable users to determine with confidence the identity of a person or device at a distance, not least by establishing the authenticity of a given document.

1.4 PROJECT SYNOPSIS

Our final year project can be categorized into two parts as follows:

1) Encryption of two 25 bit binary data streams using a 2x2 encryption matrix where each element is an irreducible polynomial of degree 6.This results in two polynomials of degree 6 which represent the encrypted forms of the two original 25 bit binary data streams.

2) Decryption of the encrypted forms using a 2x2 decryption matrix which is the inverse of the encryption matrix. This results in the two original 25 bit binary data streams.

The entire project is accomplished using ‘C’ programming. The reason for opting for ‘C’ programming language is its simplicity and ease of interpretation.

All the operations on the irreducible polynomials are accomplished in GF (13^n) fields. To take our project to the next higher level, we have opted for GF (13^n) fields instead of GF (2^n) fields. We have introduced GF(13^n) fields since a lot of research and works have been done in GF(2^n) fields already.

2

MATHEMATICS OF CRYPTOGRAPHY

2.1 MODULAR ARITHMETIC

In mathematics, modular arithmetic (sometimes called modulo arithmetic or clock arithmetic) is a system of arithmetic for integers, where numbers "wrap around" after they reach a certain value — the modulus. Modular arithmetic was introduced by Carl Friedrich Gauss in his book Disquisitiones Arithmeticae, published in 1801.

2.1.1 The Congruence Relation

Modular arithmetic can be handled mathematically by introducing a congruence relation on the integers that is compatible with the operations of the ring of integers: addition, subtraction, and multiplication. For a fixed modulus *n*, it is defined as follows.

Two integers *a* and *b* are said to be congruent modulo *n*, if their difference $a - b$ is an integer multiple of *n*. If this is the case, it is expressed as:

$$a \equiv b \pmod{n}.$$

The above statement is read: "*a* is congruent to *b* modulo *n*".

For example,

$$38 \equiv 14 \pmod{12}$$

because $38 - 14 = 24$, which is a multiple of 12. For positive *n* and non-negative *a* and *b*, congruence of *a* and *b* can also be thought of as asserting that these two numbers have the same remainder after dividing by the modulus *n*. So,

$$38 \equiv 2 \pmod{12}$$

because, when divided by 12, both numbers have the same remainder, .1666... (38/12 = 3.166..., 2/12 = .1666...). From the prior definition we also see that their difference, a - b = 36, is a whole number (integer) multiple of 12 (n = 12, 36/12 = 3).

The same rule holds for negative values of *a*:

$$-3 \equiv 2 \pmod{5}.$$

A remark on the notation: Because it is common to consider several congruence relations for different moduli at the same time, the modulus is incorporated in the notation. In spite of the ternary notation, the congruence relation for a given modulus is binary. This would have been clearer if the notation $a \equiv_n b$ had been used, instead of the common traditional notation.

The properties that make this relation a congruence relation (respecting addition, subtraction, and multiplication) are the following.

If $a_1 \equiv b_1 \pmod{n}$ and $a_2 \equiv b_2 \pmod{n}$, then:

- $(a_1 + a_2) \equiv (b_1 + b_2) \pmod{n}$
- $(a_1 - a_2) \equiv (b_1 - b_2) \pmod{n}$
- $(a_1 a_2) \equiv (b_1 b_2) \pmod{n}$.

2.1.2 Applications

Modular arithmetic is referenced in number theory, group theory, ring theory, knot theory, abstract algebra, cryptography, computer science, chemistry and the visual and musical arts.

It is one of the foundations of number theory, touching on almost every aspect of its study, and provides key examples for group theory, ring theory and abstract algebra.

In cryptography, modular arithmetic directly underpins public key systems such as RSA and Diffie-Hellman, as well as providing finite fields which underlie elliptic curves, and is used in a variety of symmetric key algorithms including AES, IDEA, and RC4.

In computer science, modular arithmetic is often applied in bitwise operations and other operations involving fixed-width, cyclic data structures. The modulo operation, as implemented in many programming languages and calculators, is an application of modular arithmetic that is often used in this context.

In chemistry, the last digit of the CAS registry number (a number which is unique for each chemical compound) is a check digit, which is calculated by taking the last digit of the first two parts of the CAS registry number times 1, the next digit times 2, the next digit times 3 etc., adding all these up and computing the sum modulo 10.

In music, arithmetic modulo 12 is used in the consideration of the system of twelve-tone equal temperament, where octave and enharmonic equivalency occurs (that is, pitches in a 1∶2 or 2∶1 ratio are equivalent, and C-sharp is considered the same as D-flat).

The method of casting out nines offers a quick check of decimal arithmetic computations performed by hand. It is based on modular arithmetic modulo 9, and specifically on the crucial property that $10 \equiv 1 \pmod 9$.

More generally, modular arithmetic also has application in disciplines such as law (see e.g., apportionment), economics, (see e.g., game theory) and other areas of the social sciences, where proportional division and allocation of resources plays a central part of the analysis.

2.2 ALGEBRAIC STRUCTURES

Cryptography requires sets of integers and specific operations that are defined for those sets. The combination of the set and the operations that are applied to the elements of the set is called an algebraic structure.

Three common algebraic structures are defined as follows:

1)GROUPS: A group (G) is a set of elements with a binary operation (•) that satisfies four properties (or axioms). A commutative group,also called an abelian group is a group in which the operator satisfies an extra property, commutativity.The four properties for groups plus commutativity are defined as follows:

a)Closure: If a and b are elements of G, then c=a•b is also an element of G.

b)Associativity:If a,b and c are elements of G,then (a•b) •c=a•(b•c).

c)Commutativity:For all a and b in G,we have a•b=b•a.

d)Existence of identity:For all a in G,there exists an element e,called the identity element,such that e•a=a•e=a.

e)Existence of inverse: For all a in G,there exists an element i,called the inverse of a,such that a•i=a•i=e.

2)RING: A ring, R = <{…}, •, >, is an algebraic structure with two operations.The 1^{st} operation must satisfy all five properties required for an abelian group.The second operation must satisfy only the 1^{st} two.In addition the 2^{nd} operation must be distributed over the 1^{st}.

3)FIELD: A field, denoted by F = <{…}, •, > is a commutative ring in which the second operation satisfies all five properties defined for the first operation except that the identity of the first operation has no inverse.

Galois showed that for a field to be finite, the number of elements should be p^n, where p is a prime and n is a positive integer. The finite fields are generally called ***Galois fields*** and denoted as ***GF(p^n).***

The finite fields are classified as follows:

- The order , or number of elements, of a finite field is of the form p^n, where p is a prime number called the characteristic of the field, and n is a positive integer.
- For every prime number p and positive integer n, there exists a finite field with p^n elements.
- Any two finite fields with the same number of elements are isomorphic. That is, under some renaming of the elements of one of these, both its addition and multiplication tables become identical to the corresponding tables of the other one.

GF(*p*) is simply the ring of integers modulo *p*. That is, one can perform operations (addition, subtraction, multiplication) using the usual operation on integers, followed by reduction modulo *p*. For instance, in GF(5), 4+3=7 is reduced to 2 modulo 5. Division is multiplication by the inverse modulo *p*, which may be computed using the extended Euclidean algorithm.

Finite fields are important in number theory, algebraic geometry, Galois theory, cryptography, and coding theory.

2.3 POLYNOMIALS

Although we can directly define the rules for addition and multiplication operations on n-bit words that satisfy the properties in GF(13^n) field,it is easier to work with a representation of n-bit words,a polynomial of degree n-1.

A polynomial of degree $n - 1$ is an expression of the form

$$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1x^1 + a_0x^0$$

where x^i is called the ith term and a_i is called coefficient of the *i*th term.

Although we are familiar with polynomials in algebra, to represent a n-bit word by a polynomial we need to follow some rules:

1) The power of x defines the position of the bit in the n-bit word. This means the left most bit is at position zero (related to x^0); the rightmost bit is at position n-1(related to x^{n-1}).

2) Since we define all operations in GF (13^n) field, the polynomial coefficients can be any integer between 0 and 12.

2.3.1 Irreducible Polynomials

Before defining the operations on polynomials, we need to discuss about the modulus polynomials. Addition of two polynomials never creates a polynomial out of the set. However, multiplication of two polynomials may create a polynomial with degree more than n-1.This means we need to divide the result by a modulus keep only the remainder.

For the set of polynomials in GF (p^n), a group of polynomials of degree n is defined as the modulus. The modulus in this case acts as a prime polynomial, which means that no polynomials in the set can divide this polynomial. A prime polynomial cannot be factored into a polynomial with degree of less than n. Such polynomials are referred to as irreducible polynomials.

For each, there is often more than one irreducible polynomial, which means when we define our GF (p^n) we need to declare which irreducible polynomial we are using as the modulus.

PROGRAM-1:

This is a program to find out whether a polynomial of degree less than equal to 10 is irreducible or not. This program provides the factors if it is reducible.

Algorithm:

i) The degree of the polynomial is inputted and the coefficients of different powers of x are stored in a 11 element array with other elements initialized to zero.

ii) The array is send to func1 () which then evaluates the value of polynomial and checks if it is zero. If not it displays the polynomial is irreducible.If it is zero,it passes the value to div() function for division.

iii) The div() function uses a null array which is dynamically declared and uses it for finding out the remainder of 1st step of division and sends it back to func1().

iv) This process occurs recursively until we get a linear polynomial and the factorization is complete.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<alloc.h>
void div(int,int,int *);
void func1(int *,int);
void func1(int *b,int m)
{ int i,k=0,j;
 long int p=0;
        for(i=1;i<=12;i++)
        {p=0;
        for(j=m;j>=0;j--)
        p=(b[j]*pow(i,j))+p;
        if(p%13==0)
        {k++;
        break;
        }
        }
        if(k==0)
        {if(m>0)
        printf("%d",i);
        printf("The polynomial is irreducible");
        }
```

```
else
        {printf("%d\n",i);
        printf("A factor is:(x+%d)\n",13-i);
        div(m-1,13-i,b);
        }}
void div(int p,int i,int *b)
{int *a=NULL,k,s;
a=(int *)calloc((p+1),2);
a[p]=1;
for(s=p;s>0;s--)
{k=*(b+s)-(i*(*(a+s)));
if(k<0)
{for(int j=0;k<0;j++)
k=k+13;
}
*(a+s-1)=k;
}
func1(a,p);
}
void main()
{int b[11]={0,0,0,0,0,0,0,0,0,0,0},n;
clrscr();
printf("Enter the highest power of x<10:");
scanf("%d",&n);
printf("Enter the values of coefficients of powers of x with coeff of
highest power of x=1 and rest<13:\n");
for(int i=n;i>=0;i--)
scanf("%d",&b[i]);
func1(b,n);
getch();
}
```

OUTPUT:

Enter the highest power of x<10: 7

Enter the values of coefficients of powers of x with coefficient of highest power of x=1 and rest<13:

1 3 5 7 9 8 6 4

The polynomial is irreducible.

Enter the highest power of x<10: 3

Enter the values of coefficients of powers of x with coefficient of highest power of x=1 and rest<13:

1 3 5 10

The polynomial is reducible.

A factor is: (x+7)

A factor is: (x+5)

A factor is: (x+4)

Note: The polynomial coefficients are written in decreasing power of x.

2.3.2 Operations on irreducible polynomials

ADDITION AND SUBTRACTION: Addition is very easy. It involves the addition of the coefficients of the corresponding terms in $GF(13^n)$.Addition of two polynomials of degree n-1 always create a polynomial with degree n-1, which means we do not need to reduce the result using the modulus.

Additive identity: The additive identity in a polynomial is a zero polynomial (polynomial with all coefficients set to zero).

Additive inverse: The additive inverse of a polynomial with coefficients in $GF(13^n)$ is the polynomial itself.

Addition and subtraction operations on polynomials are the same operations.

MULTIPLICATION: Multiplication in polynomials is the sum of the multiplication of each term of the 1^{st} polynomial with each term of the 2^{nd} polynomial.

1. The coefficient multiplication is done in GF (13^n).
2. The multiplying x^i by x^j results in x^{i+j}.
3. The multiplication may create terms with degree more than $n - 1$, which means the result needs to be reduced using a modulus polynomial.

DIVISION: Polynomial division is an algorithm for dividing a polynomial by another polynomial of the same or lower degree resulting in quotient and remainder polynomials.

PROGRAM-2:

This program performs division of two polynomials of any degree such that degree of the dividend polynomial is greater than or equal to the degree of the divisor polynomial.

Algorithm:

i) This program uses dynamic memory allocation. It takes the size of the dividend and the divisor during the runtime and then generates coefficient arrays dynamically.
ii) The main program 1st inputs the size of dividend and divisor polynomials.
iii) According to the input values it generates arrays for the coefficients of dividend, divisor, quotient and remainder i.e. allocates memory for them dynamically.
iv) It then inputs the coefficients of the dividend and the divisor from the user and stores them in the respective arrays.
v) It then follows the normal algorithm for division as discussed earlier to find out the quotient and remainder arrays or polynomials.

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
void main()
{ int m,n,i,j,k;
clrscr();
printf("\nEnter the degree of dividend and divisor:\n");
scanf("%d %d",&n,&m);
```

```
int * b= (int *) calloc(n,2);
int * e= (int *) calloc(n-m,2);
int * d= (int *) calloc(m,2);
int * a= (int *) calloc(m,2);
a[0]=1;
b[0]=1;
printf("\nEnter the coefficients of divident(coeff of highest degree of x 1 taken):\n");
for(i=1;i<=n;i++)
scanf("%d",&b[i]);
for(i=0;i<=n;i++)
printf(" %d",b[i]);
printf("\nEnter the coefficients of divisor(coeff of highest degree of x 1 taken):\n");
for(i=1;i<=m;i++)
scanf("%d",&a[i]);
for(i=0;i<=m;i++)
printf(" %d",a[i]);
for(i=0;i<=m;i++)
d[i]=b[i];
e[0]=d[0];
for(i=1;i<=n-m;i++)
{ for(j=1;j<=m;j++)
{ d[j-1]=d[j]-(a[j]*e[i-1]);
if(d[j-1]<0)
{for(k=0;d[j-1]<0;k++)
d[j-1]=d[j-1]+13;}}
d[m]=b[i+m];
e[i]=d[0];
}
```

```
printf("\nThe coefficients of quotient are :\n");
for(i=0;i<=(n-m);i++)
printf("%d\t",e[i]);
printf("\nThe coefficients of remainder are :\n");
for(i=1;i<=m;i++)
printf("%d\t",d[i]);
getch();
}
```

OUTPUT:

```
Enter the degrees of dividend and divisor:  8     5

Enter the coefficients of dividend:
        2     4     6     8     9     5     4     3     7
Enter the coefficients of divisor:
        5     7     9     3     4     5

The coefficients of quotient are:
        3     7     12    12
The coefficients of remainder are:
        5     0     2     12    12
```

```
Enter the degrees of dividend and divisor:  12    7

Enter the coefficients of dividend:
        9     11    4     0     2     5     7     8     5     6     2     4
Enter the coefficients of divisor:
        1     2     4     5     7     9     3     2
```

The coefficients of quotient are:

9 9 12 3 7 6

The coefficients of remainder are:

5 8 4 10 3 9 5

Note: The polynomial coefficients are written in decreasing power of x.

MULTIPLICATIVE INVERSE: The extended Euclidean algorithm can also be used to calculate the multiplicative inverse of a polynomial in a finite field.

The inverse of a polynomial p(x) in the modulus of another polynomial d(x) in $GF(13^n)$ is defined as a polynomial k(x) of degree same as that of p(x) such that its product with p(x) in modulus of d(x) leaves remainder 1.

This can be better defined as

$$(p(x) * k(x)) \bmod d(x) = 1$$

Here d(x) needs to have degree greater than both p(x) and k(x).

PROCEDURE:

For finding out the inverse of a polynomial in $GF(13^n)$ by Euclidean method we have to define 5 new entities namely

q(x) - the quotient of the division

r(x) - the remainder of the division

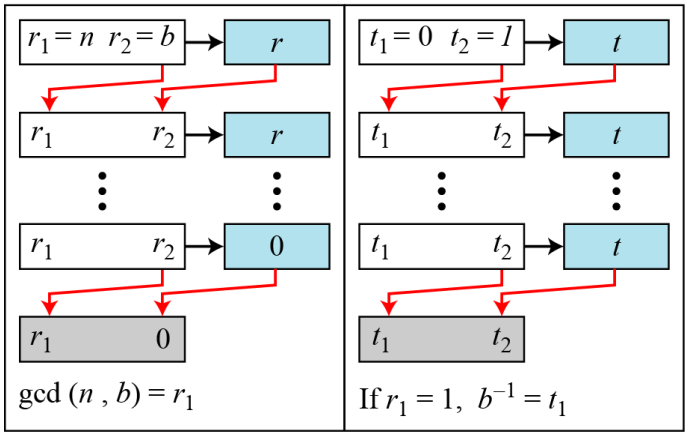
t1(x), t2(x) – two polynomials that are operated along with p(x) and q(x)

t(x) -the result polynomial

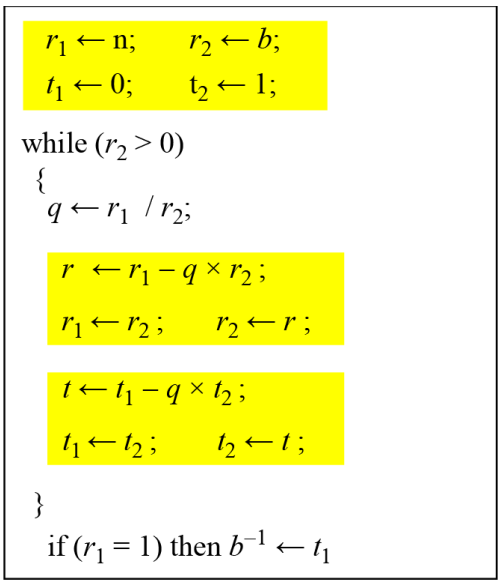
We store the quotient and remainder of the division of d(x) by p(x) in q(x) and r(x). Initially, the polynomials t1(x) and t2 (x) have values 0 and 1 respectively.
The value of t(x) is calculated by

$$t(x)=t1(x) - (t2(x)*q(x))$$

These are the values stored in each polynomial after the 1st iteration. In the 2nd iteration, p(x) shifts to d(x) , r(x) shifts to p(x), t2(x) shifts to t1(x) and t(x) shifts to t2(x).The same procedure is followed again as described above.The iterations are repeated until we get a 0 in r(x).The value at d(x) is the GCF of actual p(x) and d(x).And we get the inverse by dividing the value at t1(x) by the value at d(x). The polynomial thus obtained is the inverse of the polynomial p(x) in modulus of d(x).



a. Process



b. Algorithm

Here r1 and r2 are d(x) and p(x).

Example:

q(x)	d(x)	p(x)	r(x)	t1(x)	t2(x)	t(x)
$7x+10$	$x^7+2x^6+4x^5+5x^4+7x^3+9x^2+3x+2$	$2x^6+3x^5+5x^4+7x^3+9x^2+6x+1$	$4x^5+10x^4+4x^3+7x^2+1x+5$	0	1	$6x+3$
$7x+6$	$2x^6+3x^5+5x^4+7x^3+9x^2+6x+1$	$4x^5+10x^4+4x^3+7x^2+1x+5$	$8x^4+12x^3+12x^2+4x+10$	1	$6x+3$	$10x^2+8x+9$
$7x+7$	$4x^5+10x^4+4x^3+7x^2+1x+5$	$8x^4+12x^3+12x^2+4x+10$	$5x^3+12x^2+7x+0$	$6x+3$	$10x^2+8x+9$	$8x^3+4x^2+4x+5$
$12x+10$	$8x^4+12x^3+12x^2+4x+10$	$5x^3+12x^2+7x+0$	$3x^2+12x+10$	$10x^2+8x+9$	$8x^3+4x^2+4x+5$	$8x^4+2x^3+12x+11$
$6x+6$	$5x^3+12x^2+7x+0$	$3x^2+12x+10$	$5x+5$	$8x^3+4x^2+4x+5$	$8x^4+2x^3+12x+11$	$4x^5+5x^4+9x^3+10x^2+9x+4$
$11x+7$	$3x^2+12x+10$	$5x+5$	1	$8x^4+2x^3+12x+11$	$4x^5+5x^4+9x^3+10x^2+9x+4$	$8x^6+8x^5+4x^4+11x^3+9x+9$
$5x+5$	$5x+5$	1	0	$4x^5+5x^4+9x^3+10x^2+9x+4$	$8x^6+8x^5+4x^4+11x^3+9x+9$	0
	1	0		$8x^6+8x^5+4x^4+11x^3+9x+9$	0	

Hence we get the inverse of $(2x^6+3x^5+5x^4+7x^3+9x^2+6x+1)$ in modulo $(x^7+2x^6+4x^5+5x^4+7x^3+9x^2+3x+2)$ is $(8x^6+8x^5+4x^4+11x^3+9x+9)$

PROGRAM-3:

This program gives the multiplicative inverse of a polynomial in the modulo of another polynomial.

Algorithm:

In this program we are finding out the inverse of any degree 6 polynomial in modulo of ($x^7+2x^6+4x^5+5x^4+7x^3+9x^2+3x+2$). But we can also make the program to input the coefficients of this polynomial on whose modulo we are finding out the inverse. The steps involved in finding out the inverse are:

i)This program uses 3 functions namely:

Rec- this function accepts the value (1-12) through its argument and returns its inverse in modulo 13.

Mul - this function gives the product of 2 polynomial, one of degree 6 and the other of degree 1 in GF(13).

Anydiv - this function divides 2 polynomials to give the remainder and quotient.

ii)Initially, we initialize the values of the entities used for finding out the inverse i.e. d(x),r(x),t1(x),t2(x),t(x).

iii) Then we input the coefficients of the polynomial (p(x))whose inverse we have to find out.

iv) Then we divide d(x) by p(x) to obtain r(x) and q(x). Then we use the value of q(x) to find out the value of t(x) as

$$t(x)= (t1(x)-(t2(x) * q(x)))$$

v) We shift p(x) to d(x), r(x) to p(x), t2(x) to t1(x), and t(x) to t2(x).

vi) Steps 4 and 5 are repeated until we get 0 in place of p(x) such that the value in place of d(x) is the GCF of p(x) and d(x).

vi)We get the inverse of original polynomial p(x) as the polynomial at t1(x) divided by value at d(x) which is the GCF of the 2 polynomials. Thus the inverse is obtained.

```
#include<stdio.h>
#include<conio.h>
int rec(int a)
{int b[13]={0,1,7,9,10,8,11,2,5,3,4,6,12};
return (b[a]);}
void mul(int *a,int *b,int *c)
{int k,m;
for(int i=0;i<=6;i++)
{for(int j=0;j<=1;j++)
{k=i+j;
switch(k)
{case 0:
 c[0]=a[i]*b[j];
 c[0]=c[0]%13;
 break;
case 1:
 c[1]=c[1]+(a[i]*b[j]);
 c[1]=c[1]%13;
 break;
case 2:
 c[2]=c[2]+(a[i]*b[j]);
 c[2]=c[2]%13;
 break;
case 3:
 c[3]=c[3]+(a[i]*b[j]);
 c[3]=c[3]%13;
 break;
```

```
case 4:
 c[4]=c[4]+(a[i]*b[j]);
 c[4]=c[4]%13;
 break;
case 5:
 c[5]=c[5]+(a[i]*b[j]);
 c[5]=c[5]%13;
 break;
case 6:
 c[6]=c[6]+(a[i]*b[j]);
 c[6]=c[6]%13;
 break;
case 7:
 c[7]=c[7]+(a[i]*b[j]);
 c[7]=c[7]%13;
 break;
}}}
}
void anydiv(int *a,int *b,int *r,int *q,int c)
{int n,i,j,k;
int e[6],d[7];
for(i=0;i<=6-c;i++)
{d[i]=b[i];
}
e[0]=(d[0]*rec(a[0]))%13;
for(i=1;i<=2;i++)
{ for(j=1;j<=6-c;j++)
{ d[j-1]=d[j]-(a[j]*e[i-1]);
if(d[j-1]<0)
{for(k=0;d[j-1]<0;k++)
d[j-1]=d[j-1]+13;
}
if(d[j-1]>13)
d[j-1]=d[j-1]%13;
}
```

```
d[6-c]=b[i+6-c];
e[i]=(d[0]*rec(a[0]))%13;
}
for(n=0;n<=6-c;n++)
{r[n]=d[n];
}
for(n=0;n<=1;n++)
{q[n]=e[n];
}
}
void main()
{int
a[7],b[8]={1,2,4,5,7,9,3,2},q[2]={0,0},r[7]={0,0,0,0,0,0,0},t1[7]={0,0,0
,0,0,0,0},t2[7]={0,0,0,0,0,0,1},t[8],temp[8]={0,0,0,0,0,0,0,0},p[8],inv[7]
;
clrscr();
printf("\nEnter the coefficients of polynomial:\n");
for(int i=0;i<=6;i++)
scanf("%d",&a[i]);
printf("\n");
for(i=0;i<=6;i++)
printf("%d\t",a[i]);
for(i=0;i<=5;i++)
{anydiv(a,b,r,q,i);
temp[0]=0;
for(int j=0;j<=6;j++)
temp[j+1]=t1[j];
for(j=0;j<=7;j++)
p[j]=0;
mul(t2,q,p);
for(j=0;j<=7;j++)
{t[j]=temp[j]-p[j];
for(int k=0;t[j]<0;k++)
t[j]=t[j]+13;
}
```

```
for(j=0;j<=6-i;j++)
b[j]=a[j];
for(j=0;j<=5-i;j++)
a[j]=r[j];
for(j=0;j<=6;j++)
{t1[j]=t2[j];
t2[j]=t[j+1];}}
printf("\n\ngcf= %d and its reciprocal= %d\n",a[0],rec(a[0]));
printf("\n And the inverse is:\n");
for(i=0;i<=6;i++)
{inv[i]=rec(a[0])*t2[i];
if(inv[i]>=13)
inv[i]=inv[i]%13;
printf("%d\t",inv[i]);
}getch();}
```

OUTPUT:

Enter the coefficients of polynomial of degree 6:

1 3 5 9 2 6

gcf =2 and its reciprocal =7

And the inverse is:

12 12 8 4 12 11 7

Note: The polynomial coefficients are written in decreasing power of x.

3

ENCRYPTION

3.1 INTRODUCTION

Our method of cryptography involves the use of a 2X2 encryption matrix of randomly generated degree 6 irreducible polynomials in GF (13^n) and all the operations are carried out in the modulus of a degree 7 irreducible polynomial which can again be varied during each time the encryption is carried on a set of data. The program inputs binary data stream, carries out modulo 13 operations on the corresponding polynomial in GF (13^n) and returns back the encrypted binary data stream.

The encryption technique implemented in the project can be considered more or less equivalent to Hill Cipher technique.

Hill Cipher:

It is an example of a polyalphabetical cipher invented by Lester S.Hill.In this case, the plaintext is divided into equal size blocks. The blocks are encrypted one at a time in such a way that each character in the block contributes to the encryption of other characters in the block. For this Hill cipher belongs to a category of ciphers called block ciphers.

In a Hill cipher, the key is a square matrix of size m x n in which m is the size of the block. If we call the key matrix K, then

$$\mathrm{K} = \begin{bmatrix} k_{11} & k_{12} & \cdots & k_{1m} \\ k_{21} & k_{22} & \cdots & k_{2m} \\ \vdots & \vdots & & \vdots \\ k_{\mathrm{m1}} & k_{\mathrm{m2}} & \cdots & k_{mm} \end{bmatrix}$$

Let us show how one block of the ciphertext is encrypted. If we call the m characters in the plaintext block $P_1, P_2, P_3, \ldots\ldots\ldots, Pm$, the corresponding characters in the ciphertext block are $C_1, C_2, C_3, \ldots\ldots\ldots, C_m$, then we have

$$\begin{aligned} \mathrm{C}_1 &= \mathrm{P}_1 k_{11} + \mathrm{P}_2 k_{21} + \cdots + \mathrm{P}_m k_{m1} \\ \mathrm{C}_2 &= \mathrm{P}_1 k_{12} + \mathrm{P}_2 k_{22} + \cdots + \mathrm{P}_m k_{m2} \\ &\cdots \\ \mathrm{C}_\mathrm{m} &= \mathrm{P}_1 k_{1m} + \mathrm{P}_2 k_{2m} + \cdots + \mathrm{P}_m k_{mm} \end{aligned}$$

Note: The key matrix in the Hill cipher needs to have a multiplicative inverse.

3.2 PROCEDURE

Let A be the input data matrix with

$$A= [a11 \quad a21]^T$$

where, a11,a21- the 2 GF(13) polynomials obtained after converting the 2 binary streams into polynomials.

Let B be the encryption matrix used to carry out the encryption process. Then B is defined as

$$B=\begin{bmatrix} b11 & b12 \\ b21 & b22 \end{bmatrix}$$

Where b11, b12, b21, b22 are the 4 degree 6 irreducible polynomials used to define the encryption matrix.

Now let C be the product matrix obtained by multiplying the input data matrix with the encryption matrix.

Then C can be given as

$$C= B \text{ X } A$$

And let C be defined as

$$C= [c11 \quad c21]^T$$

where c11,c21 are the 2 degree 12 polynomials obtained by multiplying the elements of matrices B and A.

Now let D (n) be the degree 7 irreducible polynomial on whose modulus the encryption and decryption are carried out.

Dividing the elements of C by D(n) we get the remainder matrix R such that

$$R= [r11 \quad r21]^T$$

Where r11 and r21 are obtained as the remainders of the division of c11 and c21 by D(n).

These 2 remainder polynomials are then again converted to binary format to obtain the encrypted data.

3.3 ALGORITHM

The algorithm governing this process is as follows:-

1. This program uses 4 functions namely-
    a. **B213-** this function converts 25 bit binary data stream into corresponding GF(13) polynomial.
    b. **Anydiv** – this function divides 2 polynomial to give the remainder and quotient. This program returns only the remainder to the calling function.
    c. **Revbin** – this function converts the GF(13) polynomial into binary data stream which is now the encrypted data stream.
    d. **Mul** – this function gives the product of 2 degree 6 polynomial in GF(13).
2. Initially, in the main program we initialize the encryption matrix by defining the 4 irreducible polynomial of degree 6 and the

degree 7 irreducible polynomial on whose modulo we carry out the complete procedure.

3. The **b213** function is called twice to input 2 25-bit binary data stream and convert them to GF(13) polynomial. All the operations for encryption will be then carried out in this polynomial.
4. Now the **MUL** function is called 4 times to get the product matrix obtained by multiplication of encryption matrix (2X2) with input matrix(2X1).
5. The two polynomial elements of product matrix are divided by the divisor polynomial D(n) through **anydiv** function to obtain the remainder matrix of 2 degree 6 polynomial(2X1). This is the encrypted data in GF(13).
6. The 2 remainder polynomials of the remainder matrix are send to **revbin** function to obtain the respective 25-bit encrypted binary data stream.

3.4 PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<alloc.h>
void b213(int *);
void anydiv(int *,int *,int *);
void revbin(int *);
void mul(int *a,int *b,int *c)
{int k,m;
for(int i=0;i<=6;i++)
{for(int j=0;j<=6;j++)
{k=i+j;
```

```
switch(k)
{case 0:
 c[0]=a[i]*b[j];
 c[0]=c[0]%13;
 break;
case 1:
 c[1]=c[1]+(a[i]*b[j]);
 c[1]=c[1]%13;
 break;
case 2:
 c[2]=c[2]+(a[i]*b[j]);
 c[2]=c[2]%13;
 break;
case 3:
 c[3]=c[3]+(a[i]*b[j]);
 c[3]=c[3]%13;
 break;
case 4:
 c[4]=c[4]+(a[i]*b[j]);
 c[4]=c[4]%13;
 break;
case 5:
 c[5]=c[5]+(a[i]*b[j]);
 c[5]=c[5]%13;
 break;
case 6:
 c[6]=c[6]+(a[i]*b[j]);
 c[6]=c[6]%13;
 break;
case 7:
 c[7]=c[7]+(a[i]*b[j]);
 c[7]=c[7]%13;
 break;
case 8:
 c[8]=c[8]+(a[i]*b[j]);
```

```
 c[8]=c[8]%13;
 break;
case 9:
 c[9]=c[9]+(a[i]*b[j]);
 c[9]=c[9]%13;
 break;
case 10:
 c[10]=c[10]+(a[i]*b[j]);
 c[10]=c[10]%13;
 break;
case 11:
 c[11]=c[11]+(a[i]*b[j]);
 c[11]=c[11]%13;
 break;
case 12:
 c[12]=c[12]+(a[i]*b[j]);
 c[12]=c[12]%13;
 break;
}}}
}
void b213(int *c)
{
int a[25],i;
long int b=0,d;
printf("\nEnter the 25 bit binary stream:\n");
for(i=0;i<=24;i++)
{scanf("%d",&a[i]);
if(a[i]!=0 && a[i]!=1)
{printf("Re-enter the value of bit(0 or 1):\n");
i--;
}
}
for(i=0;i<=24;i++)
b=b+(pow(2,i)*a[i]);
printf("\nThe decimal value of bit stream is:\n %ld",b);
```

```
d=b;
for(i=6;i>=0;i--)
{c[i]=d%13;
d=d/13;
}
printf("\n\nThe polynomial in mod 13 is:\n %d(x6)+ %d(x5)+ %d(x4)+
%d(x3)+ %d(x2)+ %d(x)+ %d",c[0],c[1],c[2],c[3],c[4],c[5],c[6]);
}
void anydiv(int *a,int *b,int *c)
{int m,n,i,j,k;
int e[6],d[8];
for(i=0;i<=7;i++)
d[i]=b[i];
e[0]=d[0];
for(i=1;i<=6;i++)
{ for(j=1;j<=7;j++)
{ d[j-1]=d[j]-(a[j]*e[i-1]);
if(d[j-1]<0)
{for(k=0;d[j-1]<0;k++)
d[j-1]=d[j-1]+13;
}
}
d[7]=b[i+7];
e[i]=d[0];
}
for(n=0,m=6;n<=6;n++,m--)
{c[m]=d[n];
}
}
void revbin(int *a)
{ int c[25],i;
long int b=0,d;
for(i=0;i<=6;i++)
b=b+(pow(13,i)*a[i]);
printf("\nThe decimal value of polynomial is:\n %ld",b);
```

```
d=b;
for(i=24;i>=0;i--)
{c[i]=d%2;
d=d/2;
}
printf("\nThe encrypted binary stream is:\n");
for(i=24;i>=0;i--)
printf("%d  ",c[i]);
}
void main()
{ int i,j,a11[7],a21[7],c11[13],c12[13],c22[13],c21[13],c[13];
clrscr();
int b11[7]={1,3,5,2,1,7,1};
int b12[7]={1,5,2,8,7,4,9};
int b21[7]={1,7,8,9,6,5,4};
int b22[7]={1,5,4,2,7,8,2};
int d[8]={1,2,4,5,7,9,3,2};
int r1[7]={0,0,0,0,0,0,0};
int r2[7]={0,0,0,0,0,0,0};
for(i=0;i<=12;i++)
{c11[i]=0;
c12[i]=0;
c21[i]=0;
c22[i]=0;}
b213(a11);
b213(a21);
mul(b11,a11,c11);
mul(b12,a21,c12);
mul(b21,a11,c21);
mul(b22,a21,c22);
for(i=0;i<=12;i++)
{c11[i]=c11[i]+c12[i];
 c11[i]=c11[i]%13;
 c21[i]=c21[i]+c22[i];
 c21[i]=c21[i]%13;
```

```
}
printf("\nThe two matrices are:\n 1st \t2nd");
for(i=0;i<=12;i++)
printf("\n %d \t%d",c11[i],c21[i]);
anydiv(d,c11,r1);
anydiv(d,c21,r2);
printf("\n The 2 remainders are:\n 1st \t2nd");
for(i=6;i>=0;i--)
printf("\n %d \t%d",r1[i],r2[i]);
revbin(r1);
revbin(r2);
getch();
}
```

OUTPUT:

Enter the 25 bit binary stream (LSB 1st):
1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1

The decimal value is: 17764111

The polynomial in mod 13 is:
$3(x^6) + 8(x^5) + 10(x^4) + 12(x^3) + 8(x^2) + 1(x) + 1$

Enter the 25 bit binary stream (LSB 1st):
1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1

The decimal value is: 20132659

The polynomial in mod 13 is:
$4(x^6) + 2(x^5) + 2(x^4) + 11(x^3) + 9(x^2) + 2(x) + 1$

The two matrices are:

7	0	4	2	4	2	0	12	5	7	8	4	10
7	12	1	2	2	5	9	5	3	5	6	8	6

The two remainders are:

10	8	12	7	11	10	12
10	5	4	1	3	8	6

The decimal value of remainder is: 51598546

The encrypted binary stream is:

0 1 0 0 1 0 1 1 0 0 1 0 1 0 1 0 1 1 0 0 1 0 0 0 1

The decimal value of remainder is: 50241613

The encrypted binary stream is:

1 0 1 1 0 0 1 0 0 0 0 0 0 1 0 1 0 1 1 1 1 1 1 1 0

Note: The polynomial coefficients are written in decreasing power of x.

4

DECRYPTION

4.1 INTRODUCTION

Decryption is the process of getting back the original data or the plain text from the cipher text. In this method we obtain the encrypted binary data stream, convert it back to GF(13) polynomials and carry out exactly the reverse process to obtain the original data stream. In this process we multiply the inverse of encryption matrix used during the encryption process with the encrypted data matrix to obtain the output data matrix having useful data. All the operations in this process are also carried out in modulus of the same degree 7 irreducible polynomial. The data is again converted back to binary form from polynomials of GF(13).

4.2 PROCEDURE

If B is the encryption matrix and R is the matrix obtained after converting the received encrypted data stream into GF(13) polynomials then the fundamental process involved in decryption is

$$I = B^{-1} X R$$

But to find out B^{-1} we need to find out determinant of B , adjoint of B and then find out their product in modulus of D(n) to get the B^{-1} .

We have, determinant of B given as

$$\Delta = ((b11 * b22) – (b21 * b12))$$

But this is again a degree 12 polynomial. To convert it back to degree 6 polynomial we find out its modulus with D(n).

Next we need to find out its inverse. We do this using Euclidean method in the modulus of D(n).

Then we have to find out the adjoint of B. The adjoint of B is defined as

$$Adj(B) = \begin{bmatrix} b22 & -b12 \\ -b21 & b11 \end{bmatrix}$$

So inverse of B is given by

$$B^{-1} = (1/\Delta) * adj(B)$$

Again the elements of the matrix thus obtained will be of order 12. To convert them back to polynomial of degree 6 we have to take their modulo with D(n).

Now to change the encrypted data stream into readable form we have to get the product of the inverse encryption matrix and data matrix which have the 2 encrypted polynomials.

$$I = B^{-1} X R$$

But the matrix thus obtained has its elements again in degree 12. To convert it back to degree 6 we have to take the modulus with D(n).

The matrix thus obtained has the 2 polynomials which when converted to binary format will produce the original data streams required.

4.3 ALGORITHM

The algorithm governing this process is as follows-

1. This program uses 6 functions namely-
    a. **Rec-** this function accepts the value through its argument and returns its inverse in modulo 13
    b. **Revbin -** this function converts the GF(13) polynomial into binary data stream which is now the decrypted data stream.
    c. **B213 -** this function converts 25 bit binary data stream into corresponding GF(13) polynomial.
    d. **Anydiv -** this function divides 2 polynomial to give the remainder and quotient. This program returns only the remainder to the calling function.

e. **Mul -** this function gives the product of 2 degree 6 polynomial in GF(13).
f. **Pinv –** this function takes the value of a degree 6 polynomial and returns its inverse in modulo D(n) by Euclidean method.

2. Initially we call the **b213** function to input the 2 25 bit encrypted data streams and convert them to GF(13) polynomials.
3. Then we find the determinant of B calling **mul** twice to get (b11 * b22) and (b12 * b21) and then subtracting the 2^{nd} from 1^{st} .the degree 12 polynomial thus obtained is then converted to degree 6 by modulo D(n).
4. The **pinv** function is then called to find out the inverse of det(B).
5. Then we find out the adjoint of B and multiply it with inverse of its determinant to obtain B^{-1} .
6. After obtaining B^{-1} we then multiply the data matrix obtained in step 2 with the B^{-1} to get the decrypted data matrix.
7. The matrix thus obtained has its elements of degree 12. They are converted to degree 6 by taking modulo of D(n).
8. These polynomials thus obtained are then send to **revbin** function to obtain the decrypted binary data streams.

4.4 PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int rec(int a)
{int b[13]={0,1,7,9,10,8,11,2,5,3,4,6,12};
return (b[a]);
}
void revbin(int *a)
```

```
{ int c[25],i;
long int b=0,d;
for(i=0;i<=6;i++)
b=b+(pow(13,i)*a[i]);
printf("\nThe decimal value of polynomial is:\n %ld",b);
d=b;
for(i=24;i>=0;i--)
{c[i]=d%2;
d=d/2;
}
printf("\nThe original binary stream is:\n");
for(i=24;i>=0;i--)
printf("%d  ",c[i]);
}

void anydiv1(int *b,int *d)
{int e[6];
static int a[8]={1,2,4,5,7,9,3,2};
for(int i=0;i<=7;i++)
d[i]=b[i];
e[0]=(d[0]*rec(a[0]))%13;
for(i=1;i<=6;i++)
{ for(int j=1;j<=7;j++)
{ d[j-1]=d[j]-(a[j]*e[i-1]);
if(d[j-1]<0)
{for(int k=0;d[j-1]<0;k++)
d[j-1]=d[j-1]+13;
}
}
d[7]=b[i+7];
e[i]=(d[0]*rec(a[0]))%13;

}
}
void b213(int *c)
```

```
{
int a[25],i;
long int b=0,d;
printf("\nEnter the 25 bit binary stream:\n");
for(i=0;i<=24;i++)
{scanf("%d",&a[i]);
if(a[i]!=0 && a[i]!=1)
{printf("Re-enter the value of bit(0 or 1):\n");
i--;
}
}
for(i=0;i<=24;i++)
b=b+(pow(2,i)*a[i]);
printf("\nThe decimal value of bit stream is:\n %ld",b);
d=b;
for(i=6;i>=0;i--)
{c[i]=d%13;
d=d/13;
}
printf("\n\nThe polynomial in mod 13 is:\n %d(x6)+ %d(x5)+ %d(x4)+
%d(x3)+ %d(x2)+ %d(x)+ %d",c[0],c[1],c[2],c[3],c[4],c[5],c[6]);
}
void mul1(int *a,int *b,int *c)
{int k,m;
for(int i=0;i<=6;i++)
{for(int j=0;j<=1;j++)
{k=i+j;
switch(k)
{case 0:
 c[0]=a[i]*b[j];
 c[0]=c[0]%13;
 break;
case 1:
 c[1]=c[1]+(a[i]*b[j]);
 c[1]=c[1]%13;
```

```
 break;
case 2:
 c[2]=c[2]+(a[i]*b[j]);
 c[2]=c[2]%13;
 break;
case 3:
 c[3]=c[3]+(a[i]*b[j]);
 c[3]=c[3]%13;
 break;
case 4:
 c[4]=c[4]+(a[i]*b[j]);
 c[4]=c[4]%13;
 break;
case 5:
 c[5]=c[5]+(a[i]*b[j]);
 c[5]=c[5]%13;
 break;
case 6:
 c[6]=c[6]+(a[i]*b[j]);
 c[6]=c[6]%13;
 break;
case 7:
 c[7]=c[7]+(a[i]*b[j]);
 c[7]=c[7]%13;
 break;
}}}
}
void mul(int *a,int *b,int *c)
{int k,m;
for(int i=0;i<=6;i++)
{for(int j=0;j<=6;j++)
{k=i+j;
switch(k)
{case 0:
 c[0]=a[i]*b[j];
```

```
 c[0]=c[0]%13;
 break;
case 1:
 c[1]=c[1]+(a[i]*b[j]);
 c[1]=c[1]%13;
 break;
case 2:
 c[2]=c[2]+(a[i]*b[j]);
 c[2]=c[2]%13;
 break;
case 3:
 c[3]=c[3]+(a[i]*b[j]);
 c[3]=c[3]%13;
 break;
case 4:
 c[4]=c[4]+(a[i]*b[j]);
 c[4]=c[4]%13;
 break;
case 5:
 c[5]=c[5]+(a[i]*b[j]);
 c[5]=c[5]%13;
 break;
case 6:
 c[6]=c[6]+(a[i]*b[j]);
 c[6]=c[6]%13;
 break;
case 7:
 c[7]=c[7]+(a[i]*b[j]);
 c[7]=c[7]%13;
 break;
case 8:
 c[8]=c[8]+(a[i]*b[j]);
 c[8]=c[8]%13;
 break;
case 9:
```

```
 c[9]=c[9]+(a[i]*b[j]);
 c[9]=c[9]%13;
 break;
case 10:
 c[10]=c[10]+(a[i]*b[j]);
 c[10]=c[10]%13;
 break;
case 11:
 c[11]=c[11]+(a[i]*b[j]);
 c[11]=c[11]%13;
 break;
case 12:
 c[12]=c[12]+(a[i]*b[j]);
 c[12]=c[12]%13;
 break;
}}}
}
void anydiv(int *a,int *b,int *r,int *q,int c)
{int n,i,j,k;
int e[6],d[7];
for(i=0;i<=6-c;i++)
{d[i]=b[i];
}
e[0]=(d[0]*rec(a[0]))%13;
for(i=1;i<=2;i++)
{ for(j=1;j<=6-c;j++)
{ d[j-1]=d[j]-(a[j]*e[i-1]);
if(d[j-1]<0)
{for(k=0;d[j-1]<0;k++)
d[j-1]=d[j-1]+13;
}
if(d[j-1]>13)
d[j-1]=d[j-1]%13;
}
d[6-c]=b[i+6-c];
```

```
e[i]=(d[0]*rec(a[0]))%13;
}
for(n=0;n<=6-c;n++)
{r[n]=d[n];
}
for(n=0;n<=1;n++)
{q[n]=e[n];
}
}
void pinv(int *a,int *inv)
{int
b[8]={1,2,4,5,7,9,3,2},q[2]={0,0},r[7]={0,0,0,0,0,0,0},t1[7]={0,0,0,0,0,
0,0},t2[7]={0,0,0,0,0,0,1},t[8],temp[8]={0,0,0,0,0,0,0,0},p[8];
for(int i=0;i<=5;i++)
        {anydiv(a,b,r,q,i);
        temp[0]=0;
        for(int j=0;j<=6;j++)
        temp[j+1]=t1[j];
        for(j=0;j<=7;j++)
        p[j]=0;
        mul1(t2,q,p);
        for(j=0;j<=7;j++)
        {t[j]=temp[j]-p[j];
        for(int k=0;t[j]<0;k++)
        t[j]=t[j]+13;
        }
        for(j=0;j<=6-i;j++)
        b[j]=a[j];
        for(j=0;j<=5-i;j++)
        a[j]=r[j];
        for(j=0;j<=6;j++)
        {t1[j]=t2[j];
        t2[j]=t[j+1];
        }
        }
```

```
for(i=0;i<=6;i++)
{inv[i]=rec(a[0])*t2[i];
if(inv[i]>=13)
inv[i]=inv[i]%13;
}
}
void main()
{
int
a1[7],a2[7],r[7],r1[7],c[7],k[13],c1[13]={0,0,0,0,0,0,0,0,0,0,0,0,0},c2[13
]={0,0,0,0,0,0,0,0,0,0,0,0,0},d1[7],d2[7];
int b11[7]={1,3,5,2,1,7,1};
int b12[7]={1,5,2,8,7,4,9};
int b21[7]={1,7,8,9,6,5,4};
int b22[7]={1,5,4,2,7,8,2};
int r11[7],r12[7],r21[7],r22[7];
clrscr();
b213(a1);
b213(a2);
mul(b11,b22,c1);
mul(b12,b21,c2);
for(int i=0;i<=12;i++)
{c1[i]=c1[i]-c2[i];
 if(c1[i]>=13)
 c1[i]=c1[i]%13;
 if(c1[i]<0)
 for(int j=0;c1[i]<0;j++)
 c1[i]=c1[i]+13;
 }
anydiv1(c1,r);
pinv(r,r1);
printf("\nThe inverse of determinant of encryption matrix is:\n");
for(i=0;i<=6;i++)
printf("%d  ",r1[i]);
for(i=0;i<=6;i++)
```

```
{c[i]=b11[i];
b11[i]=b22[i];
b22[i]=c[i];
b12[i]=13-b12[i];
b21[i]=13-b21[i];
}
printf("\nThe adjoint of encryption matrix:\nb11 \tb12  \tb21  \tb22" );
for(i=0;i<=6;i++)
printf("\n%d  \t%d  \t%d  \t%d ",b11[i],b12[i],b21[i],b22[i]);
printf("\nThe inverse of the encryption matrix is:\n");
for(i=0;i<=12;i++)
c2[i]=0;
mul(b11,r1,c2);
anydiv1(c2,r11);
printf("\nr11");
for(i=0;i<=6;i++)
printf("\t%d",r11[i]);
for(i=0;i<=12;i++)
c2[i]=0;
mul(b12,r1,c2);
anydiv1(c2,r12);
printf("\nr12");
for(i=0;i<=6;i++)
printf("\t%d",r12[i]);
for(i=0;i<=12;i++)
c2[i]=0;
mul(b21,r1,c2);
anydiv1(c2,r21);
printf("\nr21");
for(i=0;i<=6;i++)
printf("\t%d",r21[i]);
for(i=0;i<=12;i++)
c2[i]=0;
mul(b22,r1,c2);
anydiv1(c2,r22);
```

```
printf("\nr22");
for(i=0;i<=6;i++)
printf("\t%d",r22[i]);
for(i=0;i<=12;i++)
c1[i]=0;
mul(b11,a1,c1);
for(i=0;i<=12;i++)
c2[i]=0;
mul(b12,a2,c2);
printf("\n");
for(i=0;i<=12;i++)
{c1[i]=c1[i]+c2[i];
if(c1[i]>13)
c1[i]=c1[i]%13;
if(c1[i]<0)
for(int j=0;c1[i]<0;j++)
c1[i]=c1[i]+13;
k[i]=c1[i];
}
anydiv1(c1,d1);
for(i=0;i<=12;i++)
c1[i]=0;
mul(b21,a1,c1);
for(i=0;i<=12;i++)
c2[i]=0;
mul(b22,a2,c2);
printf("\n");
for(i=0;i<=12;i++)
{c1[i]=c1[i]+c2[i];
if(c1[i]>13)
c1[i]=c1[i]%13;
if(c1[i]<0)
for(int j=0;c1[i]<0;j++)
c1[i]=c1[i]+13;
}
```

```
anydiv1(c1,d2);
printf("\nThe product matrix is:\n");
for(i=0;i<=12;i++)
printf("%d\t",k[i]);
printf("\n");
for(i=0;i<=12;i++)
printf("%d\t",c1[i]);
printf("\nThe 2 remainders are:\n");
for(i=0;i<=6;i++)
printf("%d\t",d1[i]);
printf("\n");
for(i=0;i<=6;i++)
printf("%d\t",d2[i]);
revbin(d1);
revbin(d2);
getch();
}
```

OUTPUT:

The encrypted binary stream is:
0 1 0 0 1 0 1 1 0 0 1 0 1 0 1 0 1 1 0 0 1 0 0 0 1

The decimal value of remainder is: 51598546

The polynomial in mod 13 is:
$10(x^6) + 8(x^5) + 12(x^4) + 7(x^3) + 11(x^2) + 10(x) + 12$

The encrypted binary stream is:
1 0 1 1 0 0 1 0 0 0 0 0 0 1 0 1 0 1 1 1 1 1 1 1 0

The decimal value of remainder is: 50241613

The polynomial in mod 13 is:
$10(x^6) + 5(x^5) + 4(x^4) + 1(x^3) + 3(x^2) + 8(x) + 6$

The inverse of determinant of encryption matrix:

4	2	11	1	9	2	6

Adjoint of encryption matrix:

b11	1	5	4	2	7	8	2
b12	12	8	11	5	6	9	4
b21	12	6	5	4	7	8	9
b22	1	3	5	2	1	7	1

Inverse of the encryption matrix:

r11	4	9	7	3	2	8	0
r12	0	12	12	5	1	8	7
r21	9	6	6	4	1	0	6
r22	3	2	9	9	11	0	4

Product matrix:

1	8	6	3	5	5	3	0	5	5	0	5	3
3	11	3	5	2	7	7	9	6	12	0	1	5

The two remainders are:

3	8	10	12	8	1	1
4	2	2	11	9	2	1

The decimal value is: 17764111
The original binary stream (LSB 1st) is:
1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 1

The decimal value is: 20132659
The original binary stream (LSB 1st) is:
1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1

Note: The polynomial coefficients are written in decreasing power of x.

5

CONCLUSION

5.1 UNIQUENESS OF THE PROJECT

The cryptographic technique implemented in our project has an edge over other techniques in the following respect:

1. This technique provides excellent security to the input data. This is because decryption is not possible without prior knowledge of the encryption matrix and the divisor polynomial.
2. Since, the encryption matrix and the divisor polynomial are generated randomly; these can be changed every time we implement the encryption technique. Thus, making anonymous decryption impossible.
3. Computation is quiet easy which makes easy hardware implementation.
4. Implementation using ‘C’ programming language makes it quite lucid and interpretable.

5.2 SCOPE FOR FUTURE WORK

A self invertible matrix is a matrix which has an inverse equal to its own. It is preferable to use a self invertible matrix as an encryption matrix as the decryption process will become much easier as the job of finding out the inverse of the encryption matrix will be removed. The decryption will again require multiplication of the encryption matrix with the encrypted data matrix to get the original data matrix.

So, during encryption we have

$$R = B \times A$$

And again during the decryption we have

$$I = B \times R = A$$

Thus, the hardware implementation will become much easier and the process will become much faster.

Let A be a self invertible matrix, with

$$A = \begin{bmatrix} a11 & a12 \\ a21 & a22 \end{bmatrix}$$

Then for the matrix to be self invertible we must have

$$\Delta = -1$$

and $a11 = -a22$

Example-1: (degree-2 polynomials)

$$A_1 = \begin{bmatrix} 4x^2+6x+2 & x^2+4x+4 \\ x^2+3x+3 & 9x^2+7x+11 \end{bmatrix}$$

In this case, we take the modulus polynomial to be x^3+3x^2+5x+7.

$\Delta = -1(=12 \text{ in GF } (13))$

Example-2: (degree-3 polynomials)

$$A_2 = \begin{bmatrix} 2x^3+3x^2+5x+7 & x^3+10x^2+7x+6 \\ x^3+12x^2+6x+8 & 11x^3+10x^2+8x+6 \end{bmatrix}$$

In this case, we take the modulus polynomial to be $x^4+7x^3+6x^2+11x+2$.

$\Delta = -1(=12 \text{ in GF } (13))$

The research on (3X3), (4X4), (5X5) self invertible matrices is going on the use of which will make the process much faster along with high data security. In future we may see techniques utilizing matrices of much higher dimensions but encrypting and decrypting at a much faster rate as compared to present techniques.

REFERENCES

[1] W.Stallings; “Cryptography and Network Security” 2nd Edition, Prentice Hall, 1999
[2] Bruce Schneir: Applied Cryptography, 2nd edition, John Wiley & Sons, 1996
[3] Cryptography and Network Security – Behrouz A. Forouzan

[4] http://en.wikipedia.org/wiki/Cryptography
[5] http://en.wikipedia.org/wiki/Finite_field_arithmetic
[6] http://en.wikipedia.org/wiki/Extended_Euclidean_algorithm
[7] http://en.wikipedia.org/wiki/Modular_arithmetic
[8] http://www.pgpi.org/doc/pgpintro/
[9] http://en.wikipedia.org/wiki/Cryptography
[10] http://www.esat.kuleuven.be/cosic/intro/